Efficient Virtual Machine Consolidation Approach Based on User Inactivity Detection

Jan Fesl
Institute of Applied Informatics
Faculty of Science
University of South Bohemia
České Budějovice, Czech Republic
email: jfesl@prf.jcu.cz

Vineet Gokhale
Institute of Applied Informatics
Faculty of Science
University of South Bohemia
České Budějovice, Czech Republic
email: vgokhale@prf.jcu.cz

Marie Feslová
Institute of Applied Informatics
Faculty of Science
University of South Bohemia
České Budějovice, Czech Republic
email: dolezm05@prf.jcu.cz

***Abstract*—Large cloud architectures consist of numerous high-performance servers, each hosting a multitude of Virtual Machines (VMs). Naturally, the server resources for processing and storage are shared among VMs, which, in turn, could be simultaneously accessed by several authorized users. Resource reallocation takes place after a session terminates. However, failure of systematic session termination causes blockage of resources resulting in severe under-utilization. In order to mitigate such scenarios, one needs to efficiently detect user inactivity for timely release of the resources. This is a non-trivial task. To this end, we propose a hybrid resource-desktop monitoring technique, which involves capturing of user interaction with the client computer, in addition to monitoring the client-server network activity. The rationale behind this approach is that even in case of lightweight applications, the user interactions cause continuous changes in the visual contents being displayed. Periodic screenshots of the client screen and network activity between client and server provide crucial information about the user inactivity. Our preliminary investigation suggests that such self-organizing virtualization infrastructure is a promising direction for the design of modern cloud-based services.**

Keywords – cloud; consolidation; neural network.

I. INTRODUCTION

Cloud-based provisioning of memory and computational resources has proved to be one of the profound inventions of modern-day computer science. Such resources are typically available for usage to clients round the clock. One such example of cloud-based applications is Virtual Machine (VM). Entities known as virtualization servers host multiple VMs, hosted on, allowing multiple users to share the server resources simultaneously. Typically, the resources allocated to a VM are pre-configured by the system administrator. The degree of resource utilization of a VM depends solely on the type of applications being executed by its clients. Naturally, when the resources are underutilized it makes sense to re-allocate them to other VMs hosting more resource demanding applications. Moreover, very often, it is imperative to relocate certain VMs to another virtualization server with spare resources for optimization of power consumption. Such live migration of VMs, as well as re-allocation of resources across VMs in the same virtualization server is commonly referred to as *consolidation*.

After the completion of a virtual session, the virtualization server consolidates the resources tied to the *inactive* VMs to other VMs. However, in a vast majority of cases the human user leaves the session without properly terminating it, resulting in severe underutilization of the server resources. In order to mitigate such sub-optimal utilization of resources, one needs to monitor the utilization of VM, and be able to distinguish between durations of user activity and inactivity. In case of *active* VM, the resource configuration remains unchanged, whereas for instances of substantially long user inactivity, the VM can be characterized as inactive and subsequently subjected to the resource reallocation process.

Once a VM is characterized as inactive, it can be hibernated or powered off. Occasionally, if all the VMs in a virtualization server are relocated or powered off, the entire virtualization server itself can be hibernated. This step dramatically reduces the overall power consumption. In an earlier work, we proposed a novel consolidation technique [1]. However, literature also contains a fair volume of work in this domain; see, for example, [12]-[14].

The detection of state (active, inactive) of a VM is a non-trivial problem, since it is characterized by complex combinations of a gamut of parameters. The existing works take into account the parameters like memory and processor consumption of VM, network throughput of VM, login metadata, among others. Although the aforementioned works demonstrate that these parameters characterize the VM-state fairly well, we identify two more parameters that can be used for VM-state detection with higher efficiency and reliability - VM screenshot variations and VM-client network traffic profile. In this paper, we propose a novel scheme for reliable detection of VM-state that integrates the proposed new parameters with the existing ones in order to efficiently detect the current VM-state with high reliability. Our scheme treats VM as a black box, in the sense that no virtual machine introspection [2] needs to be done. All parameters used for VM-state detection are retrieved directly from hypervisors or external network devices. This simplifies the design considerations significantly.

The organization of the paper is as follows. In Section II, we provide an overview of the existing literature relevant to our work. In Section III, we describe in detail our approach for characterization of the VM state, and in Section IV, we explain our proposed approach for inactivity detection of VMs. We present preliminary measurements in Section V, and finally state our conclusions in Section VI.

II. RELATED WORK

VM-state detection is a topic that is actively being researched in the recent past; see, for example, [3]-[6]. Literature suggests that there exist primarily three main approaches of VM-state detection as follows.

A. Utilization-based approach: This approach is the most intuitive of all. The VM monitor, known as hypervisor, is able

 ISBN: 978-1-61208-703-0

to provide certain information on the run-time behavior of a VM. The VM characteristics (VMC) are mostly represented as 4-D vector of items.

VMC = {CPU utilization [%], memory utilization [MB], network throughput [Mbit/s], and I/O operation count [number]}.

The existing works use a subset of VMC parameters for VM-state detection. Solution like Pulsar [7], which is a part of OpenStack-Nova, uses just only the CPU utilization. When the CPU utilization is below a specific threshold, the system marks the VM as "inactive". Another scheme [8] leverages CPU utilization, memory utilization, and network throughput for determining the state of a VM.

B. Rule-based approach: This approach means that the data center operators define some set of heuristics, which help to identify the unused virtual machine. Typical example of such approach implementation is the NetFlix service Janitor Monkey [9]. The similar approach is used in the next solution, which is called Poncho [10]. In Poncho, the rules are not defined as global, but only for a specific workload.

C. Graph-based approach: This approach has been inspired by some programming languages like Java, C# or Python. Such languages use virtual environment for the execution of byte code compiled applications. Their garbage collectors for automated memory management identify objects "to be cleaned" by examining object references. The resource dependencies are mostly represented by acyclic graphs. In many cases, standalone cloud resources – having no dependency on other cloud resources – cannot necessarily be identified as unused resources by only using resource dependencies. In other words, some VMs can cooperate with others, but this situation cannot be easily represented in the graph. Some graph-based systems are Pleco [4] and Garbo [5].

III. CHARACTERIZATION OF VM-STATE

In this section, we describe in detail the rationale behind the selection of important features for characterization of the VM-state and explain the VM-state detection metric for each feature.

A. Resource utilization of VM: Many existing works [7][8] in the literature have experimentally demonstrated that monitoring the local resources of VM like CPU and RAM usage, as well as the overall network activity can characterize the VM-state detection fairly accurately. In this work, we adopt the strategies proposed by the aforementioned prior works. Let C denote CPU usage of VM, M denote the ratio of memory currently being utilized to total memory capacity of VM. Let N denote the current network throughput related to maximal network throughput. The overall VM utilization (U) can then be expressed as shown Eq. (1).

$$U = \frac{1}{(1-C)(1-N)(1-M)} \quad (1)$$

B. Client-VM network activity: In order to optimally utilize the communication network between client and VM, any standard terminal service, like Remote Desktop Protocol (RDP) and Secure SHell (SSH), transfers only incremental information. For example, when the user continuously interacts with the VM, the contents to be displayed to the client change relentlessly, and hence large number of packets are exchanged. On the other hand, when the changes are insignificant, packets are transmitted occasionally. For instance, when the user is reading a document with negligible/less amount of mouse scrolling, the displayed contents remain predominantly unchanged. Since no significant information needs to be exchanged, the packet transmissions are sparse. On the other hand, if a multimedia file is being played at VM, irrespective of the display contents packets need to be constantly exchanged. Note that the overall network activity discussed previously subsumes the network activity between client and VM. However, it should be noted that some activities, like installation of software updates at VM that generate substantial network activity, may not be of interest from the client's perspective. Therefore, monitoring the network activity between client and VM, in addition to overall network activity, provides fine-grained network information which can be utilized in precisely characterizing the current state of VM. Terminal service activity can be expressed as follows. TR_h is an empirically set number of packets, which must be reached for flow activity detection. TR_h is an empirical value related to a specific time interval (t). The overall measurement time consists of K number t intervals. For each t interval, can be measured current count of packets C_t. If $C_t < TR_h$ then such TI is marked as "inactive". The total inactivity T [%] can be deduced as shown in Eq. (2).

$$T = \frac{\#\ of\ inactivity\ intervals}{\#\ of \text{total}\ intervals} \quad (2)$$

Necessary information about the traffic from network devices alike switches or routers can be provided via NETFLOW/IPFIX protocol, which does not affect the routers performance.

C. VM screenshot: We refer to the contents of the VM that are currently being displayed to the client. The Hypervisor creates a VM screenshot from the content of virtual graphic card memory. The idea behind using VM screenshot as an important parameter in VM-state detection is that whenever the user interacts actively with VM, the display contents keep changing continuously. Therefore, as a preliminary the differential information in consecutive screenshots can be utilized to detect the VM-state. Screenshots with same dimensions are generated periodically by the Hypervisor. Let S_k denote the k^{th} screenshot. It is important to remark that our approach considers only the differential information in consecutive screenshots, and not the actual display contents themselves. This also reduces the computational load significantly. Let D_k denote the differential information between S_{k+1} and S_k as shown in Eq. (3).

$$DS(x,y) = C(x,y) - P(x,y) \quad (3)$$

An example of differential screenshot can be seen in Figure 1. From single differential screenshots, it is possible to assemble the sequence of differential screenshots.

We created a supervised learning-based classifier which is capable of analyzing if a screenshot change indicates the activity/inactivity of the user, as shown in Figure 2. For the analysis, we used deep convolutional network, which is able automatically to take into the account the position of change - for example clock on the desktop taskbar can cause the misinterpretation.

 ISBN: 978-1-61208-703-0

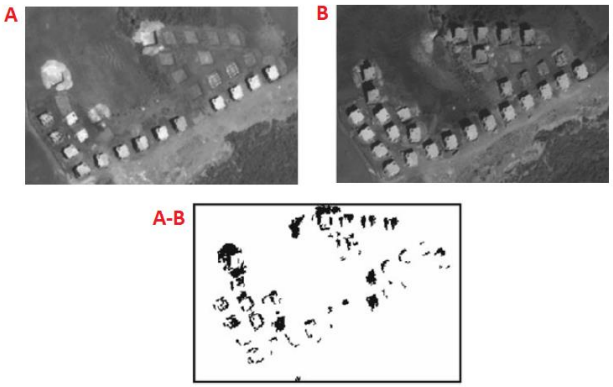

Fig. 1: Pixel differential image, which serves as an input for deep convolutional network. Such neural networks are able to detect the change and position of change as well.

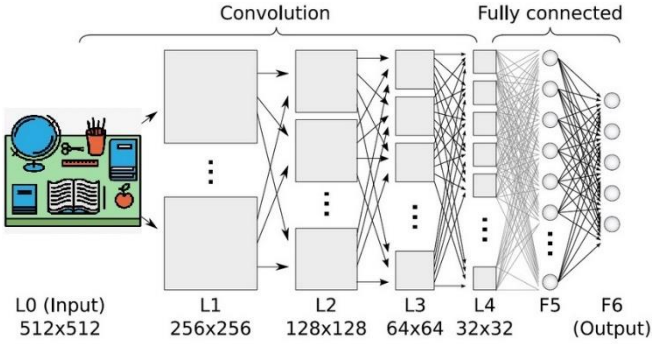

Fig. 2: Deep convolutional network used for image change classification is able to take into the account the position of change.

All differential screenshots from DS are being classified by the neural network. The result of classification is N-1 D vector of values, which expresses the probability (PI) of user inactivity for every screenshot as shown in Eq. (4).

$$PI = \{PI_1, PI_2, \ldots, PI_N\} \quad (4)$$

The average probability (P) for entire period can be computed as shown in Eq. (5).

$$P = \left(\frac{1}{N-1}\right) * \sum_{i=1}^{N-1} PI_i \quad (5)$$

P value is used in the decision three, which will be described in the next section.

IV. VM-STATE DETECTION

In this section, we present the description of the design and working principle of the proposed VM-state detection scheme.

In Figure 3, we present the architecture of a typical distributed virtualization infrastructure, along with the proposed VM-state detection elements. The clients connect to their respective VMs via a specific terminal service, like RDP or SSH. All these connections go via an IPFIX-enabled router and are monitored by a firewall. In order to relieve the VMs of monitoring the traffic stream from VM to clients, we use IPFIX [11] - a standard protocol for capturing network flows crossing the routers. The router can capture specific flow information, and subsequently report to VM Consolidator (VMC) - the principal component of the architecture. VMC coordinates with VMs and virtualization servers in order to receive vital parameters necessary for characterization of VM-state discussed in Section III. The possibility to see user screen must be supported by hypervisor.

As mentioned previously, we consider the following parameters for detection of the current state of VM: user-induced on-screen changes, network activity between client and VM, processing and memory resource consumption of VM, and overall network activity of VM. We now move to the characterization of VM-state based on the measurement of the aforementioned parameters. We propose a hierarchical VM state detection technique based on learning via ANN that classifies the current state of the VM through the aforementioned parameters using the decision tree shown in Figure 4.

Let p_n and p_{Tn} denote the decision metric and the corresponding threshold at the n^{th} stage of the decision tree. In the first stage, we take the VM screenshot variation as the evaluation parameter for decision making. On-screen visual changes refer to the changes reflecting on the VM screen when the user interacts with the virtual computer. Therefore, if on-screen changes are significantly high, then the VM is in use with higher likelihood. If $p_1 < p_{T1}$ (indicating that VM screenshot variations are significant), we treat this as the sufficient condition to infer that the VM is active. Hence, further analysis is unnecessary, thereby relieving the monitoring node (MN) of resource-heavy computations. If $p_1 \geq p_{T1}$ (indicating negligible variations), we trigger subsequent stages of analysis.

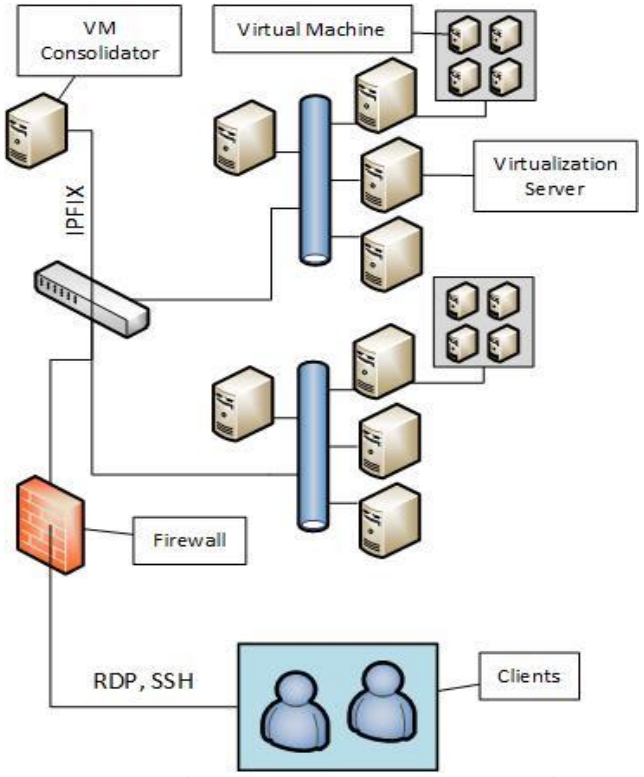

Fig. 3: Architecture of the proposed distributed virtualization infrastructure featuring clients, VMs, virtualization servers, VM consolidator, and IPFIX-enabled router.

In the second stage, we consider the client-VM network traffic profile, in addition to p_1 for characterizing the VM-state. If $p_2 < p_{T2}$ (indicating that the network activity is significant), then we infer that the VM is in active state. However, if $p_2 \geq p_{T2}$ (indicating negligible network activity), we move to the last stage of analysis.

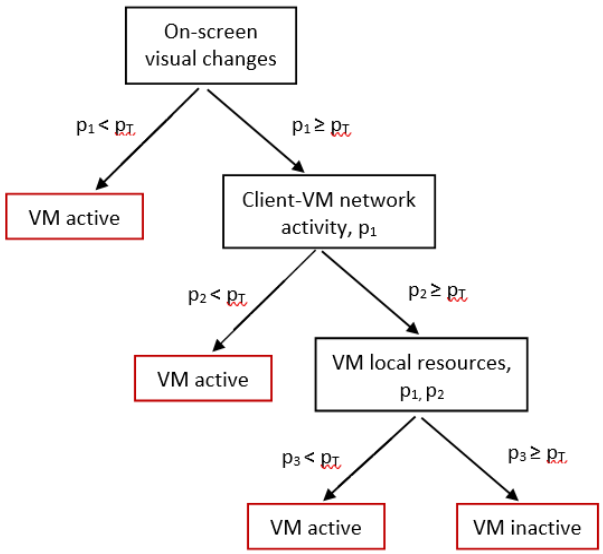

Fig. 4: Decision tree illustrating the working of proposed hierarchical VM state detection. p_n denotes the probability of VM being inactive as per the parameters considered in n^{th} stage.

In the last stage, we consider the resource utilization profile of VM for the decision making. If $p_3 < p_{T3}$ (indicating that the resource utilization is significant), then we infer that the VM is in active state. However, if $p_3 \geq p_{T3}$ (indicating negligible resource utilization), we conclude that the VM is in inactive state. This results in the hibernation of VM, and subsequent re-allocation of the resources across other VMs located in the virtualization server.

V. MEASUREMENTS

In this section, we report a preliminary measurement of packet count from VM to client for demonstrating the proof-of-concept of the proposed VM-state detection. We initiated a session between a VM and a client. In order to be able to notice the packet count variation between client and VM, we divide the user interaction into three types - *active*, *semi-active*, and *passive*. In active interaction, the user continuously performs certain tasks on VM through I/O devices. In semi-active interaction, the user only plays a multimedia content in VM, but performs no other interaction through input devices. Finally, tin passive interaction, the user simply reads a document without scrolling through it. Figure 5 presents the histogram of transmitted packets recorded during the session.

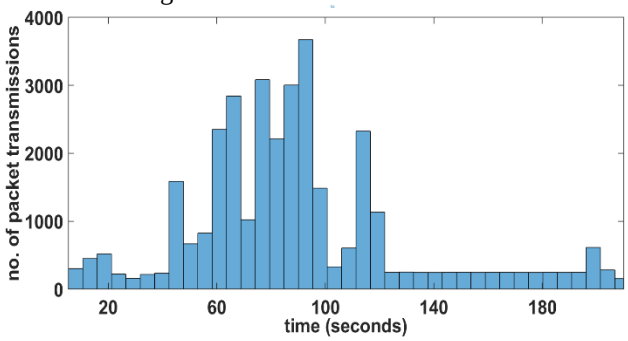

Fig. 5: Architecture of the proposed distributed virtualization infrastructure featuring clients, VMs, virtualization servers, VM consolidator, and IPFIX-enabled router.

We ask the user to perform active, semi-active, and passive interactions sequentially. In the first part of the plot, it can be seen that the packet count is significantly high indicating dense packet transmissions between VM and client during active interaction. Next, the packet transmissions are relatively sparse but non-negligible, indicating considerable traffic during semi-active interaction. Finally, the packet transmissions are near-zero indicating negligible traffic during passive interaction. This suggests that monitoring the traffic profile between VM and client could play a crucial role in accurately and efficiently determining the VM-state.

As a next step of this work, we plan to implement the proposed scheme on a real cloud infrastructure and compare its performance with state-of-the-art techniques in the domain.

VI. CONCLUSIONS

In this paper, we proposed a novel VM-state detection strategy in which two new parameters were introduced - VM screenshot and VM-client traffic profile. We described how these new parameters can improve up on the state-of-the-art techniques. We proposed the architecture of our scheme, and also proposed the usage of deep convolutional neural network for classification of VM-state. Through preliminary measurements, we demonstrated that monitoring the network activity between VM and client could lead to efficient performance. In the next phase of the project, we intend to perform extensive verification of our proposal and also improve the neural network classifier.

ACKNOWLEDGEMENT

The authors would like to thank the Technological Agency of Czech Republic for financing the current research, and Mr. Jiří Cehák for his assistance in system administration and management.

REFERENCES

[1] J. Fesl, J. Cehák, M. Doležalová, and J. Janeček, “New approach for virtual machines consolidation in heterogeneous computing systems”, International Journal of Hybrid Information Technology, vol. 9, pp. 321–332, 2016.

[2] K. Nance, B. Hay, and M. Bishop., “Virtual Machine Introspection: Observation or Interference?”, IEEE Security & Privacy, 6(5), pp. 32–37, September 2008.

[3] K. Kim, S. Zeng, Ch. Young, J. Hwang, and M. Humphre, “iCSI: A Cloud Garbage VM Collector for Addressing Inactive VMs with Machine Learning”, IEEE International Conference on Cloud Engineering, 2017.

[4] Z. Shen, Ch. C. Young, S. Zeng, K. Murthy, and K. Bai, “Identifying Resources for Cloud Garbage Collection”, 12th International Conference on Network and Service Management (CNSM), Montreal, 2016.

[5] B. Zhang, Y. Al-Dhuraibi, R. Rouvoy, F. Paraiso, and L. Seinturier “CloudGC: Recycling Idle Virtual Machines in the Cloud”, IEEE International Conference on Cloud Engineering (IC2E), Vancouver, 2017.

[6] N. Cohen and A. Bremler-Barr., “Garbo: Graph-based Cloud Resource Cleanup”, ACM Symposium on Cloud Computing (SoCC ’15), Kohala Coast, Hawaii, USA, August 2015.

[7] D. Breitgand et al. “An Adaptive Utilisation Accelerator for Virtualized Environments”, 2nd IEEE International Conference on Cloud Engineering (IC2E ’14), Boston, MA, USA, March 2014.

[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, “Black-box and Gray-box Strategies for Virtual Machine Migration”, 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’07), Cambridge, MA, USA, April 2007.

[9] Netflix, Janitor Monkey, https://github.com/Netflix/SimianArmy/wiki/Janitor-Home, available online, Jan. 2015.

[10] S. Devoid, N. Desai, and L Hochstein, “Poncho: Enabling Smart Administration of Full Private Cloud”, 27th USENIX Large Installation System Administration Conference (LISA ’13), Washington D.C., USA, November 2013.

 ISBN: 978-1-61208-703-0

[11] IPFIX protocol specification, https://tools.ietf.org/html/rfc7011, available online, Sep. 2013.

[12] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers using Reinforcement Learning", 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2014.

[13] Z. Cao and S. Dong, "An energy-aware heuristic framework for virtual machine consolidation in Cloud computing", The Journal of Supercomputing, Volume 69, Issue 1, pp 429–451, Springer, 2014.

[14] E. Feller, L. Rilling, C. Morin, "Energy-aware ant colony-based workload placement in Clouds", Proceeding GRID'11 proceedings of the 2011 IEEE/ACM 12th international conference on grid computing. IEEE Computer Society, Washington, DC, pp 26–33, 2011.

 ISBN: 978-1-61208-703-0